unit_test

July 21, 2020

1 Test Your Algorithm

1.1 Instructions

1. From the **Pulse Rate Algorithm** Notebook you can do one of the following:

- Copy over all the **Code** section to the following Code block.
- Download as a Python (`.py`) and copy the code to the following Code block.

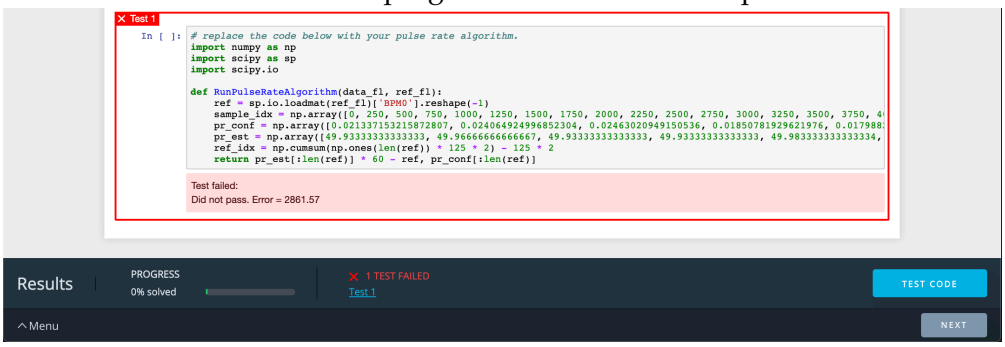
2. In the bottom right, click the Test Run button.

1.1.1 Didn't Pass

If your code didn't pass the test, go back to the previous Concept or to your local setup and continue iterating on your algorithm and try to bring your training error down before testing again.

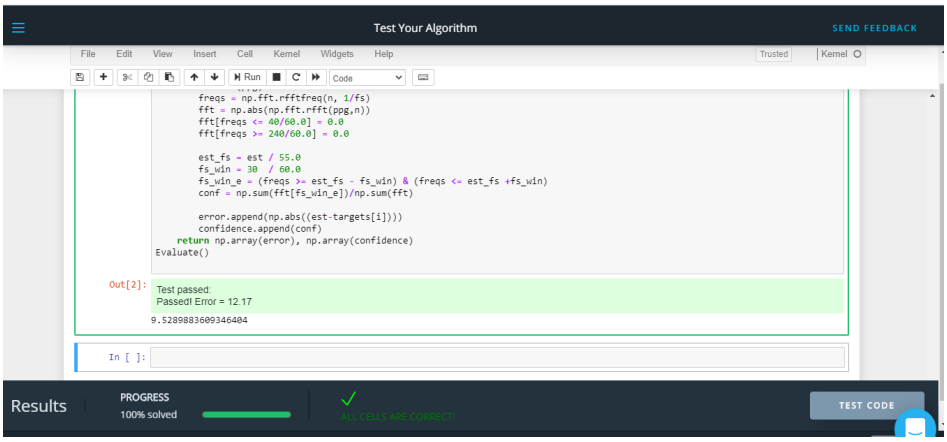
1.1.2 Pass

If your code passes the test, complete the following! You **must** include a screenshot of your code and the Test being **Passed**. Here is what the starter filler code looks like when the test is run and should be similar. A passed test will include in the notebook a green outline plus a box with **Test passed:** and in the Results bar at the bottom the progress bar will be at 100% plus a checkmark with **All cells passed**.



1. Take a screenshot of your code passing the test, make sure it is in the format `.png`. If not a `.png` image, you will have to edit the Markdown render the image after Step 3. Here is an example of what the `passed.png` would look like
2. Upload the screenshot to the same folder or directory as this jupyter notebook.

3. Rename the screenshot to `passed.png` and it should show up below.



4. Download this jupyter notebook as a `.pdf` file.
5. Continue to Part 2 of the Project.

```
In [2]: import os
        import os.path
        import glob
        import numpy as np
        import scipy as sp
        import scipy.io
        import scipy.signal
        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.model_selection import KFold
        from sklearn.metrics import mean_squared_error
        import pickle

        def LoadTroikaDataset():
            """
            Retrieve the .mat filenames for the troika dataset.

            Review the README in ./datasets/troika/ to understand the organization of the .mat f

            Returns:
                data_fls: Names of the .mat files that contain signal data
                ref_fls: Names of the .mat files that contain reference data
                <data_fls> and <ref_fls> are ordered correspondingly, so that ref_fls[5] is the
                    reference data for data_fls[5], etc...
            """
            data_dir = "./datasets/troika/training_data"
            data_fls = sorted(glob.glob(data_dir + "/DATA_*.mat"))
            ref_fls = sorted(glob.glob(data_dir + "/REF_*.mat"))
            return data_fls, ref_fls

        def LoadTroikaDataFile(data_fl):
            """
```

```
    Loads and extracts signals from a troika data file.

    Usage:
        data_fls, ref_fls = LoadTroikaDataset()
        ppg, accx, accy, accz = LoadTroikaDataFile(data_fls[0])

    Args:
        data_fl: (str) filepath to a troika .mat file.

    Returns:
        numpy arrays for ppg, accx, accy, accz signals.
    """
    data = sp.io.loadmat(data_fl)['sig']
    return data[2:]

def PreprocessTestData(data_fl, ref_fl):
    fs=125
    win_len = 8
    win_shift = 2

    sig = LoadTroikaDataFile(data_fl)
    ref = scipy.io.loadmat(ref_fl)["BPM0"]
    ref = np.array([x[0] for x in ref])
    subject_name = os.path.basename(data_fl).split('.')[0]
    start_indxs, end_indxs = get_indxs(sig.shape[1], len(ref), fs, win_len,win_shift)
    targets, features, sigs, subs = [], [], [], []
    for i, s in enumerate(start_indxs):
        start_i =  start_indxs[i]
        end_i = end_indxs[i]

        ppg = sig[0, start_i:end_i]
        accx = sig[1, start_i:end_i]
        accy = sig[2, start_i:end_i]
        accz = sig[3, start_i:end_i]

        ppg = BandpassFilter(ppg)
        accx = BandpassFilter(accx)
        accy = BandpassFilter(accy)
        accz = BandpassFilter(accz)

        feature, ppg, accx, accy, accz = Featurize(ppg, accx, accy, accz)

        sigs.append([ppg, accx, accy, accz])
        targets.append(ref[i])
        features.append(feature)
        subs.append(subject_name)

    return (np.array(targets), np.array(features), sigs, subs)
```

```
def AggregateErrorMetric(pr_errors, confidence_est):
    """
    Computes an aggregate error metric based on confidence estimates.

    Computes the MAE at 90% availability.

    Args:
        pr_errors: a numpy array of errors between pulse rate estimates and correspondin
            reference heart rates.
        confidence_est: a numpy array of confidence estimates for each pulse rate
            error.

    Returns:
        the MAE at 90% availability
    """
    # Higher confidence means a better estimate. The best 90% of the estimates
    #    are above the 10th percentile confidence.
    percentile90_confidence = np.percentile(confidence_est, 10)

    # Find the errors of the best pulse rate estimates
    best_estimates = pr_errors[confidence_est >= percentile90_confidence]

    # Return the mean absolute error
    return np.mean(np.abs(best_estimates))

def Featurize(ppg, accx, accy, accz):
    """ Create features """

    fs = 125
    n = len(ppg) * 4
    freqs = np.fft.rfftfreq(n, 1/fs)
    fft = np.abs(np.fft.rfft(ppg,n))
    fft[freqs <= 40/60.0] = 0.0
    fft[freqs >= 240/60.0] = 0.0

    acc_mag = np.sqrt(accx**2 + accy**2 + accz**2)
    acc_fft = np.abs(np.fft.rfft(acc_mag, n))
    acc_fft[freqs <= 40/60.0] = 0.0
    acc_fft[freqs >= 240/60.0] = 0.0

    ppg_feature = freqs[np.argmax(fft)]
    acc_feature = freqs[np.argmax(acc_fft)]

    return (np.array([ppg_feature, acc_feature]), ppg, accx, accy, accz)

def Evaluate():
    """
```

```
    Top-level function evaluation function.

    Runs the pulse rate algorithm on the Troika dataset and returns an aggregate error m

    Returns:
        Pulse rate error on the Troika dataset. See AggregateErrorMetric.
    """
    global reg
    reg = TrainRegressor()

    data_fls, ref_fls = LoadTroikaDataset()
    errs, confs = [], []
    for data_fl, ref_fl in zip(data_fls, ref_fls):
        errors, confidence = RunPulseRateAlgorithm(data_fl, ref_fl)
        errs.append(errors)
        confs.append(confidence)

    errs = np.hstack(errs)
    confs = np.hstack(confs)
    return AggregateErrorMetric(errs, confs)

def BandpassFilter(signal, fs = 125):
    b, a = scipy.signal.butter(3, (40/60.0, 240/60.0), btype='bandpass', fs=fs)
    return scipy.signal.filtfilt(b, a, signal)

def get_indxs(sig_len, ref_len, fs=125, win_len_s=10, win_shift_s=2):
    """
    Find start and end index to iterate over a set of signals
    """
    if ref_len < sig_len:
        n = ref_len
    else:
        n = sig_len

    start_indxs = (np.cumsum(np.ones(n) * fs * win_shift_s) - fs * win_shift_s).astype(i
    end_indxs = start_indxs + win_len_s * fs
    return (start_indxs, end_indxs)

def TrainRegressor():

    fs=125
    win_len = 8
    win_shift = 2

    data_fls, ref_fls = LoadTroikaDataset()
    targets, features, sigs, subs = [], [], [], []
    for data_fl, ref_fl in (zip(data_fls, ref_fls)):
```

```
            sig = LoadTroikaDataFile(data_fl)
            ref = scipy.io.loadmat(ref_fl)["BPM0"]
            ref = np.array([x[0] for x in ref])
            subject_name = os.path.basename(data_fl).split('.')[0]
            start_indxs, end_indxs = get_indxs(sig.shape[1], len(ref), fs, win_len,win_shift
            for i, s in enumerate(start_indxs):
                start_i =  start_indxs[i]
                end_i = end_indxs[i]

                ppg = sig[0, start_i:end_i]
                accx = sig[1, start_i:end_i]
                accy = sig[2, start_i:end_i]
                accz = sig[3, start_i:end_i]

                ppg = BandpassFilter(ppg)
                accx = BandpassFilter(accx)
                accy = BandpassFilter(accy)
                accz = BandpassFilter(accz)

                feature, ppg, accx, accy, accz = Featurize(ppg, accx, accy, accz)

                sigs.append([ppg, accx, accy, accz])
                targets.append(ref[i])
                features.append(feature)
                subs.append(subject_name)

        targets = np.array(targets)
        features = np.array(features)

        regression = RandomForestRegressor(n_estimators=400,max_depth=20)
        lf = KFold(n_splits=5)
        splits = lf.split(features,targets,subs)

        for i, (train_idx, test_idx) in enumerate(splits):
            X_train, y_train = features[train_idx], targets[train_idx]
            X_test, y_test = features[test_idx], targets[test_idx]
            regression.fit(X_train, y_train)

        return regression

    def RunPulseRateAlgorithm(data_fl, ref_fl):
        fs = 125
        win_len = 8
        win_shift = 2

        targets, features, sigs, subs = PreprocessTestData(data_fl, ref_fl)

        error, confidence = [], []
```

```
            for i,feature in enumerate(features):
                est = reg.predict(np.reshape(feature, (1, -1)))[0]
                ppg, accx, accy, accz = sigs[i]

                n = len(ppg) * 4
                freqs = np.fft.rfftfreq(n, 1/fs)
                fft = np.abs(np.fft.rfft(ppg,n))
                fft[freqs <= 40/60.0] = 0.0
                fft[freqs >= 240/60.0] = 0.0

                est_fs = est / 55.0
                fs_win = 30  / 60.0
                fs_win_e = (freqs >= est_fs - fs_win) & (freqs <= est_fs +fs_win)
                conf = np.sum(fft[fs_win_e])/np.sum(fft)

                error.append(np.abs((est-targets[i])))
                confidence.append(conf)
            return np.array(error), np.array(confidence)
        Evaluate()
```

```
Out[2]: 9.5289883609346404
```

```
In [ ]:
```